LPI exam prep: Web services

Intermediate Level Administration (LPIC-2) topic 208

Skill Level: Intermediate

David Mertz (mertz@gnosis.cx)
Developer
Gnosis Software, Inc.

25 Apr 2006

In this tutorial, the fourth in a series of seven tutorials covering intermediate network administration on Linux, David Mertz continues preparing you to take the Linux Professional Institute® Intermediate Level Administration (LPIC-2) Exam 208. Here, David Mertz discusses how to configure and run the Apache HTTP server and the Squid proxy server.

Section 1. Before you start

Learn what these tutorials can teach you and how you can get the most from them.

About this series

The Linux Professional Institute (LPI) certifies Linux system administrators at two levels: *junior level* (also called "certification level 1") and *intermediate level* (also called "certification level 2"). To attain certification level 1, you must pass exams 101 and 102; to attain certification level 2, you must pass exams 201 and 202.

developerWorks offers tutorials to help you prepare for each of the four exams. Each exam covers several topics, and each topic has a corresponding self-study tutorial on developerWorks. For LPI exam 202, the seven topics and corresponding developerWorks tutorials are:

Table 1. LPI exam 202: Tutorials and topics

LPI exam 202 topic	developerWorks tutorial	Tutorial summary
Topic 205	LPI exam 202 prep (topic	Learn how to configure a basic

	205): Networking configuration	TCP/IP network, from the hardware layer (usually Ethernet, modem, ISDN, or 802.11) through the routing of network addresses.
Topic 206	LPI exam 202 prep (topic 206): Mail and news	Learn how to use Linux as a mail server and as a news server. Learn about mail transport, local mail filtering, mailing list maintenance software, and server software for the NNTP protocol.
Topic 207	LPI exam 202 prep (topic 207): DNS	Learn how to use Linux as a DNS server, chiefly using BIND. Learn how to perform a basic BIND configuration, manage DNS zones, and secure a DNS server.
Topic 208	LPI exam 202 prep (topic 208): Web services	(This tutorial) Learn how to install and configure the Apache Web server, and learn how to implement the Squid proxy server. See detailed objectives below.
Topic 210	LPI exam 202 prep (topic 210): Network client management	Coming soon
Topic 212	LPI exam 202 prep (topic 212): System security	Coming soon
Topic 214	LPI exam 202 prep (topic 214): Network troubleshooting	Coming soon

To start preparing for certification level 1, see the developerWorks tutorials for LPI exam 101. To prepare for the other exam in certification level 2, see the developerWorks tutorials for LPI exam 201. Read more about the entire set of developerWorks LPI tutorials.

The Linux Professional Institute does not endorse any third-party exam preparation material or techniques in particular. For details, please contact info@lpi.org.

About this tutorial

Welcome to "Web services," the fourth of seven tutorials covering intermediate network administration on Linux. In this tutorial, you learn how to configure and run the Apache HTTP server and the Squid Web Proxy Cache.

As with the other tutorials in the developerWorks 201 and 202 series, this tutorial is

intended to serve as a study guide and entry point for exam preparation, rather than complete documentation on the subject. Readers are encouraged to consult LPI's detailed objectives list and to supplement the information provided here with other material as needed.

This tutorial is organized according to the LPI objectives for this topic. Very roughly, expect more questions on the exam for objectives with higher weight.

Table 2. Web services: Exam objectives covered in this tutorial

LPI exam objective	Objective weight	Objective summary
2.208.1 Implementing a Web server	Weight 2	Install and configure a Web server. This objective includes monitoring the server's load and performance, restricting client user access, configuring support for scripting languages as modules, and setting up client user authentication. Also included is the ability to configure server options to restrict usage of resources.
2.208.2 Maintaining a Web server	Weight 2	Configure a Web server to use virtual hosts, Secure Sockets Layer (SSL), and customize file access.
2.208.3 Implementing a proxy server	Weight 2	Install and configure a proxy server, including access policies, authentication, and resource usage.

Prerequisites

To get the most from this tutorial, you should already have a basic knowledge of Linux and a working Linux system on which you can practice the commands covered in this tutorial.

Section 2. About Apache and Squid

Apache Web server

Apache is the predominant Web server on the Internet as a whole; it is even more predominant among Linux servers. A few more special-purpose Web servers are available (some offering higher performance for specific tasks), but Apache is

always the default choice.

Apache comes pre-installed on most Linux distributions and is often already running after being launched during initialization, even if you have not specifically configured a Web server. If Apache is not installed, use the normal installation system of your distribution to install it, or download the latest HTTP server from the Apache HTTP Server Project. Many extra capabilities are provided by modules, many are distributed with Apache itself, and others are available from third parties.

Even though the latest Apache has been at the 2.x level since 2001, Apache 1.3.x is still in widespread use, and the 1.3.x series continues to be maintained for bug fixes and security updates. Some minor configuration differences exist between 1.3 and 2.x; a few modules are available for 1.3 that are not available for 2.x. The latest releases as of this tutorial are 1.3.34 (stable), 2.0.55 (stable), and 2.1.9 (beta).

As a rule, a new server should use the latest stable version in the 2.x series. Unless you have a specific need for an unusual older module, 2.x provides good stability, more capabilities, and overall better performance (in some tasks, such as in PHP support, 1.3 still performs better). Moving forward, new features will certainly be better supported in 2.x than in 1.3.x.

Squid proxy server

Squid is a proxy-caching server for Web clients that supports the HTTP, FTP, TLS, SSL, and HTTPS protocols. By running a cache on a local network, or at least closer to your network than the resources queried, speed can be improved and network bandwidth reduced. When the same resource is requested multiple times by machines served by the same Squid server, the resource is delivered from a server-local copy rather than requiring the request to go out over multiple network routers and to potentially slow or overload destination servers.

You can configure Squid as an explicit proxy that must be configured in each Web client (browser), or you can configure it to intercept all Web requests out of a LAN and cache all such traffic. You can configure Squid with various options regarding how long and under what conditions to keep pages cached.

Other resources

As with most Linux tools, it is always useful to examine the manpages for any utilities discussed. Versions and switches might change between utility or kernel version or with different Linux distributions. For more in-depth information, the Linux Documentation Project has a variety of useful documents, especially its HOWTOs. A variety of books on Linux networking have been published; I have found O'Reilly's *TCP/IP Network Administration*, by Craig Hunt, to be quite helpful. (See Resources later in this tutorial for links.)

Many good books have been written on working with Apache. Some are concerned with general administration, while others cover particular modules or special

configurations of Apache. Check your favorite bookseller for a range of available titles.

Section 3. Implementing a Web server

A swarm of daemons

Launching Apache is similar to launching any other daemon. Usually you want to put its launch in your system initialization scripts, but in principle you may launch Apache at any time. On most systems, the Apache server is called *httpd*, though it may be called *apache2* instead. The server is probably installed in /usr/sbin/, but other locations are possible depending on the distribution and how you installed the server.

Most of the time you will launch Apache with no options, but the `-d serverroot` and `-f config` options are worth keeping in mind. The first lets you specify a location on the local disks from where content will be served; the second lets you specify a non-default configuration file. A configuration file may override the `-f` option using the `ServerRoot` directive (most do). By default, configuration files are either apache2.conf or httpd.conf, depending on compilation options. These files might live at /etc/apache2/, /etc/apache/, /etc/httpd/conf/, /etc/httpd/apache/conf, or a few other locations depending on version, Linux distribution, and how you installed or compiled Apache. Checking `man apache2` or `man httpd` should give you system-specific details.

The Apache daemon is unusual when compared with other servers in that it usually creates several running copies of itself. The primary copy simply spawns the others, while these secondary copies service the actual incoming requests. The goal of having multiple running copies is to act as a "pool" for requests that may arrive in bundles; additional copies of the daemon are launched as needed, according to several configuration parameters. The primary copy usually runs as root, but the other copies run as a more restricted user for security reasons. For example:

Listing 1. The many faces of multiple running copies of Apache

```
# ps axu | grep apache2
root      6620     Ss   Nov12   0:00 /usr/sbin/apache2 -k start -DSSL
www-data  6621     S    Nov12   0:00 /usr/sbin/apache2 -k start -DSSL
www-data  6622     Sl   Nov12   0:00 /usr/sbin/apache2 -k start -DSSL
www-data  6624     Sl   Nov12   0:00 /usr/sbin/apache2 -k start -DSSL
dqm        313     S+   03:44   0:00 man apache2
root       637     S+   03:59   0:00 grep apache2
```

On many systems, the restricted user will be *nobody*. In Listing 1, it is `www-data`.

Including configuration files

As mentioned, the behavior of Apache is affected by directives in its configuration file. For Apache2 systems, the main configuration file is likely to reside at /etc/apache2/apache2.conf, but often this file will contain multiple `Include` statements to add configuration information from other files, possibly by wildcard patterns. Overall, an Apache configuration is likely to contain hundreds of directives and options (most not specifically documented in this tutorial).

A few files are particularly likely to be included. You might see httpd.conf for "user" settings, to utilize prior Apache 1.3 configuration files that use that name. Virtual hosts are typically specified in separate configuration files, matched on a wildcard, like in the following:

Listing 2. Specifying virtual hosts

```
# Include the virtual host configurations:
Include /etc/apache2/sites-enabled/[^.#]*
```

With Apache 2.x, modules are typically specified in separate configuration files as well (more often in the same file in 1.3.x). For example, a system of mine includes:

Listing 3. From /etc/apache2/apache2.conf

```
# Include module configuration:
Include /etc/apache2/mods-enabled/*.load
Include /etc/apache2/mods-enabled/*.conf
```

Actually using a module in a running Apache server requires two steps in the configuration file, both loading it and enabling it:

Listing 4. Loading an optional Apache module

```
# cat /etc/apache2/mods-enabled/userdir.load
LoadModule userdir_module /usr/lib/apache2/modules/mod_userdir.so
# cat /etc/apache2/mods-enabled/userdir.conf
<IfModule mod_userdir.c>
    UserDir public_html
    UserDir disabled root

    <Directory /home/*/public_html>
        AllowOverride FileInfo AuthConfig Limit
        Options MultiViews Indexes SymLinksIfOwnerMatch IncludesNoExec
    </Directory>
</IfModule>
```

The wildcards in the `Include` lines will insert all the .load and .conf files in the /etc/apache2/mods-enabled/ directory.

Notice the general pattern: Basic directives are one-line commands with some options; more complex directives nest commands using an XML-like open/close tag.

You just have to know for each directive whether it is one-line or open/close style -- you cannot choose among styles at will.

Log files

An important class of configuration directives concern logging of Apache operations. You can maintain different types of information and degrees of detail for Apache operations. Keeping an error log is always a good idea; you can specify it with the single directive:

Listing 5. Specifying an error log

```
# Global error log.
ErrorLog /var/log/apache2/error.log
```

You can customize other logs of server access, of referrers, and of other information to fit your individual setup. A logging operation is configured with two directives. First, a `LogFormat` directive uses a set of special variables to specify what goes in the log file; second, a `CustomLog` directive tells Apache to actually record events in the specified format. You can specify an unlimited number of formats regardless of whether each one is actually used. This allows you to switch logging details on and off, based on evolving needs.

Variables in a `LogFormat` are similar to shell variables, but with a leading `%`. Some variables have single letters, while others have long names surrounded by brackets, as shown in Listing 6.

Listing 6. LogFormat variables

```
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\"" combined
CustomLog /var/log/apache2/referer_log combined
```

Consult a book or full Apache documentation for the list of variables. Commonly used ones include `%h` for IP address of requesting client, `%t` for datetime of the request, `%>s` for HTTP status code, and the misspelled `%{Referer}` for the referring site that led to the served page.

The name used in the `LogFormat` and `CustomLog` directives is arbitrary. In Listing 6, the name `combined` was used, but it could just as well be `myfoobarlog`. However, a few names are commonly used and come with sample configuration files, such as `combined`, `common`, `referer`, and `agent`. These specific formats are typically supported directly by log-analyzer tools.

Section 4. Maintaining a Web server

Virtual hosts, multi-homing, and per-directory options

Individual directories served by an Apache server may have their own configuration options. However, the main configuration may limit which options can be configured locally. If per-directory configuration is desired, use the `AccessFileName` directive and typically specify the local configuration filename of .htaccess. The limitations of local configuration are specified within a `<Directory>` directive. For example:

Listing 7. Example of directory directive

```
#Let's have some Icons, shall we?
Alias /icons/ "/usr/share/apache2/icons/"
<Directory "/usr/share/apache2/icons">
      Options Indexes MultiViews
      AllowOverride None
      Order allow,deny
      Allow from all
</Directory>
```

Often working in conjunction with per-directory options, Apache can service *virtual hosts*. Multiple domain names may be served from the same Apache process, each accessing an appropriate directory. You can define virtual hosts with the `<VirtualHost>` directive; place configuration files in an included directory, such as /etc/apache2/sites-enabled/, or in a main configuration file. For example, you might specify:

Listing 8. Configuring virtual hosts

```
      <VirtualHost "foo.example.com">
          ServerAdmin webmaster@foo.example.com
          DocumentRoot /var/www/foo
          ServerName foo.example.com
          <Directory /var/www/foo>
                Options Indexes FollowSymLinks MultiViews
                AllowOverride None
                Order allow,deny
                allow from all
          </Directory>
          ScriptAlias /cgi-bin/ /usr/lib/cgi-bin/
          <Directory "/usr/lib/cgi-bin">
                AllowOverride None
                Options ExecCGI -MultiViews +SymLinksIfOwnerMatch
                Order allow,deny
                Allow from all
          </Directory>
          CustomLog /var/log/apache2/foo_access.log combined
      </VirtualHost>
      <VirtualHost "bar.example.org">
          DocumentRoot /var/www/bar
          ServerName bar.example.org
      </VirtualHost>
      <VirtualHost *>
          DocumentRoot /var/www
      </VirtualHost>
```

The final * option picks up any HTTP requests that are not directed to one of the explicitly specified names (like those addressed by IP address or addressed as an

unspecified symbolic domain that also resolves to the server machine). For virtual domains to work, DNS must define each alias with a CNAME record.

Multi-homed servers sound similar to virtual hosting, but the concept is different. Using *multi-homing*, you may specify the IP addresses to which a machine is connected in order to allow Web requests. For example, you might provide HTTP access only to the local LAN, but not to the outside world. If you specify an address to listen on, you may also indicate a non-default port. The default value for `BindAddress` is `*`, which means to accept requests on every IP address under which the server may be reached. A mixed example might look like:

Listing 9. Configuring multi-homing

```
BindAddress 192.168.2.2
Listen 192.168.2.2:8000
Listen 64.41.64.172:8080
```

In this case, we will accept all client requests from the local LAN (that use the 192.168.2.2 address) on the default port 80 and on the special port 8000. This Apache installation will also monitor client HTTP requests from the WAN address, but only on port 8080.

Limiting Web access

You may enable *per-directory* server access with the `Order`, `Allow from`, and `Deny from` commands within a `<Directory>` directive. Denied or allowed addresses may be specified by full or partial hostnames or IP addresses. `Order` lets you give precedence between the accept list and the deny list.

In many cases, you need more fine-tuned control than you can gain by simply allowing particular hosts to access your Web server. To enable user login requirements, use the `Auth*` family of commands, again within the `<Directory>` directive. For example, to set up Basic Authentication, you might use a directive as shown in Listing 10.

Listing 10. Configuring Basic Authentication

```
<Directory "/var/www/baz">
    AuthName "Baz"
    AuthType Basic
    AuthUserFile /etc/apache2/http.passwords
    AuthGroupFile /etc/apache2/http.groups
    Require john jill sally bob
</Directory>
```

You may also specify Basic Authentication within a per-directory .htaccess file. Digest Authentication is more secure than Basic, but Digest Authentication is less widely implemented in browsers. However, the weakness of Basic (that it transmits passwords in cleartext) is better solved with an SSL layer, anyway.

Support for SSL encryption of Web traffic is provided by the module `mod_ssl`. When SSL is used, data transmitted between server and client is encrypted with a dynamically negotiated password that is resistant to interception. All major browsers support SSL. For more information on configuring Apache 2.x with `mod_ssl`, see the description on the Apache Web site (see Resources for a link).

Section 5. Implementing a proxy server

Installing and running Squid

In most distributions, you can install Squid using the normal installation procedures. Get the source version of Squid from the Squid Web Proxy Cache Web site (see Resources for a link). Building from source uses the basic `./configure; make; make install` sequence.

Once installed, you may simply run it as root, /usr/sbin/squid (or whatever location your distribution uses, perhaps /usr/local/sbin/). Of course, to do much useful, you will want to configure the Squid configuration file at /etc/squid/squid.conf, /usr/local/squid/etc/squid.conf, or wherever precisely your system locates squid.conf. As with almost all daemons, you may use a different configuration file, in this case with the `-f` option.

Ports, IP addresses, http_access, and ACLs

The most important configuration options for Squid are the `http_port` options you select. You may monitor whichever ports you wish, optionally attaching each one to a particular IP address or hostname. The default port for Squid is 3128, allowing any IP address that connects to the Squid server. To cache only for a LAN, specify the local IP address instead as shown:

Listing 11. Caching Squid only for a LAN

```
# default (disabled)
# http_port 3128
# LAN only
http_port 192.168.2.2:3128
```

You may also enable caching via other Squid servers using the `icp_port` and `htcp_port`. The IPC and HTCP protocols are used for caches to communicate between themselves rather than by Web servers and clients themselves. To cache multicasts, use `mcast_groups`.

To let clients connect to your Squid server, you need to give them permission to do so. Unlike a Web server, Squid is not entirely generous with its resources. In the

simple case, we can just use a couple of subnet/netmask or CIDR (Classless Internet Domain Routing) patterns to control permissions:

Listing 12. Simple Squid access permissions

```
http_access deny 10.0.1.0/255.255.255.0
http_access allow 10.0.0.0/8
icp_access allow 10.0.0.0/8
```

You can use the `acl` directive to name access control lists (ACLs). You can name `src` ACLs that simply specify address ranges as in Listing 12, but you can also create other types of ACLs. For example:

Listing 13. Fine-tuned access permissions

```
acl mynetwork     src             192.168/16
acl asp           urlpath_regex   \.asp$
acl bad_ports     port            70 873
acl javascript    rep_mime_type -i ^application/x-javascript$
# what HTTP access to allow classes
http_access deny asp          # don't cache active server pages
http_access deny bad_ports    # don't cache gopher or rsync
http_access deny javascript   # don't cache javascript content
http_access allow mynetwork   # allow LAN everything not denied
```

Listing 13 shows only a small subset of the available ACL types. See a sample squid.conf for examples of many others. Or take a look at the Access control documentation (Chapter 6) in the Squid User's Guide (see Resources for a link).

In Listing 13, we decide not to cache URLs that end with .asp (probably dynamic content), not to cache ports 70 and 873, and not to cache returned JavaScript objects. Other than what is denied, machines on the LAN (the /16 range given) will have all their requests cached. Notice that each ACL defined has a unique, but arbitrary, name (use names that make sense; Squid does not reserve the names).

Caching modes

The simplest way to run Squid is in proxy mode. If you do this, clients must be explicitly configured to use the cache. Web browser clients have configuration screens that allow them to specify a proxy address and port rather than a direct HTTP connection. This setup makes configuring Squid very simple, but it makes clients do some setup work if they want to benefit from the Squid cache.

You can also configure Squid to run as a transparent cache. To do this, you need to either configure policy-based routing (outside of Squid itself, using `ipchains` or `ipfilter`) or use your Squid server as a gateway. Assuming you can direct external requests via the Squid server, Squid needs to be configured as follows. You may need to recompile Squid with the `--enable-ipf-transparent` option; however, in most Linux installations, this should already be fine. To configure the server for transparent caching (once it gets the redirected packets), add something like Listing 14 to your squid.conf:

Listing 14. Configuring Squid for transparent caching

```
httpd_accel_host virtual
httpd_accel_port 80
httpd_accel_with_proxy on
httpd_accel_uses_host_header on
```

Resources

Learn

- Review the entire LPI exam prep tutorial series on developerWorks to learn Linux fundamentals and prepare for system administrator certification.
- At the LPIC Program, find task lists, sample questions, and detailed objectives for the three levels of the Linux Professional Institute's Linux system administration certification.
- Get more information on configuring Apache 2.x with mod_ssl.
- Read the tutorial "Customizing Apache for maximum performance" (developerWorks, June 2002) to learn how to tweak Apache for particular environments and needs.
- The chapter on Access Control and Access Control Operators in the Squid User's Guide describes the available ACL types.
- *TCP/IP Network Administration, Third Edition* by Craig Hunt (O'Reilly, April 2002) is an excellent resource on Linux networking.
- The Linux Users Groups WorldWide home page lists 700 Linux user groups around the world. Many LUGs have local and distance study groups for LPI exams.
- The Linux Documentation Project has a variety of useful documents, especially its HOWTOs.
- In the developerWorks Linux zone, find more resources for Linux developers.
- Stay current with developerWorks technical events and Webcasts.

Get products and technologies

- Download the latest Apache Web server.
- Download Squid and additional Squid documentation.
- Order the SEK for Linux, a two-DVD set containing the latest IBM trial software for Linux from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®.
- With IBM trial software, available for download directly from developerWorks, build your next development project on Linux.

Discuss

- Participate in the discussion forum for this content.
- Check out developerWorks blogs and get involved in the developerWorks community.

About the author

David Mertz
David Mertz thinks that artificial languages are perfectly natural, but natural languages seem a bit artificial. You can reach David at mertz@gnosis.cx; you can investigate all aspects of his life at his personal Web page. Check out his book, Text Processing in Python. Suggestions and recommendations on past or future columns are welcome.

Trademarks